Design Review 2

ECE3663 Digital Integrated Circuit

Leiqing Cai (lc9ac)

Qing Qin (qq3za)

Ashley Morse (alm3tw)

4/8/2014

Introduction

The overall goal of the project is to create a DSP system that performs various arithmetic/logic operations, and to optimize the performance of the design, in terms of delay, area and power consumption. In this first stage, we built some basic components for the ALU unit, including AND, OR, and PASS. In this second stage, we competed the rest of the circuit components, including adder, subtractor, shifter, register. We also worked out the complete DSP system (excluding the arbitrary function), and verified the full functionality of the circuit.

Top-level Design

The block diagram of the ALU has been revised and shown in Figure 1. Data inputs Ain and Bin are 16-bit wide. Control inputs S_0, S_1 and S_2 are 1-bit wide. Each input connect to a register of appropriate width. Output Out is 16-bit wide, and is sourced from a register.

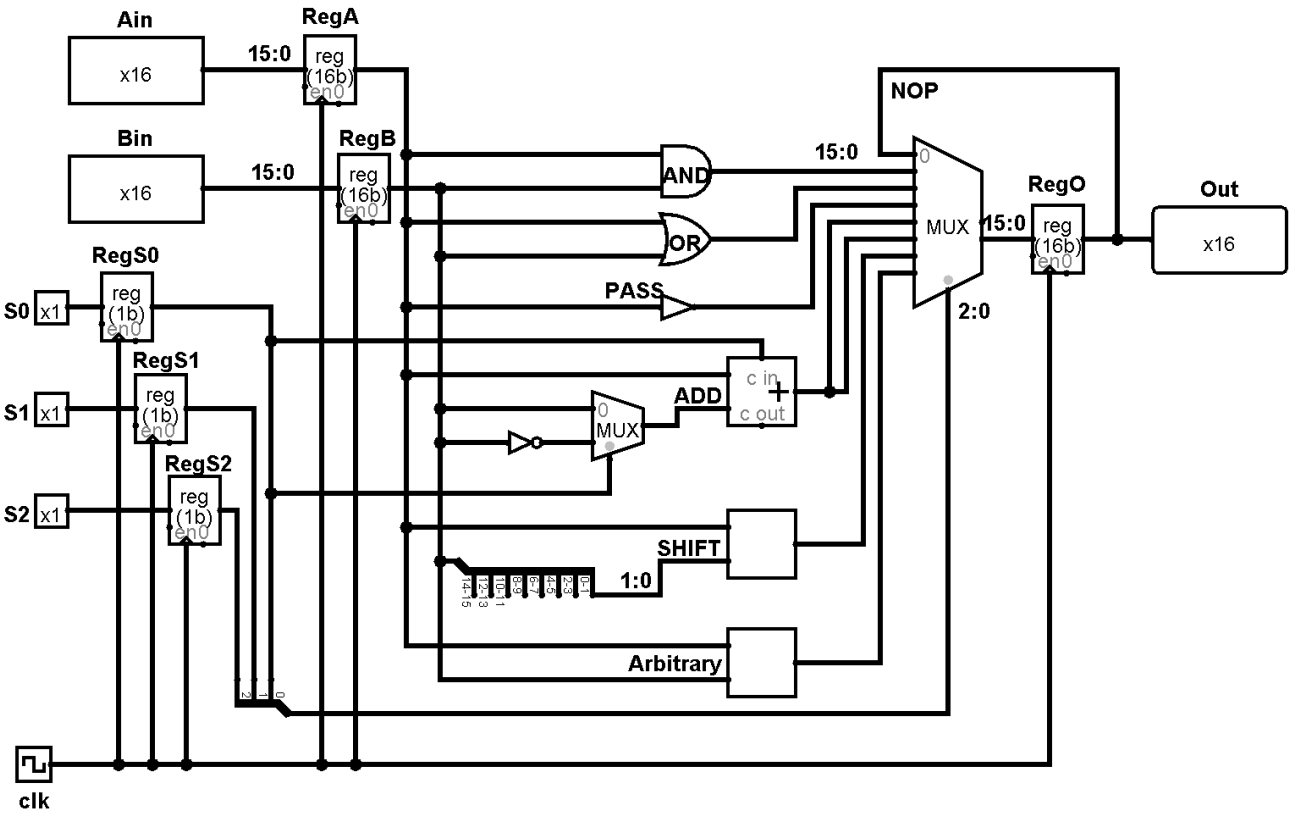

Figure 1 Revised Block Diagram of the DSP system

Function of the ALU is selected by the control bits (op-code). The selected function corresponded to each op-code is shown in Table 1. The last row of the table corresponds to the arbitrary function, which has not been implemented in the current design review.

Table 1 ALU function for different select values

S<2:0>	Function
000	NOP
001	$Out = A \,\&\, B$
010	$Out = A \mid B$
011	$Out = A$
100	$Out = A + B$
101	$Out = A - B$
110	$Out = A \ll B\langle 2{:}0\rangle$
111	N/A

Components

Introduction

Our team chose to design the system by directly editing netlists. Components including transistor wrappers, inverter, buffer, 16-bit AND gate, 16-bit OR gate, 16-bit PASS A gate, 1-bit 2:1 Mux and 1-bit 8:1 Mux has been shown in the design review 1.

Transmission Gate

One of the basic components in addition to the ones used in design review 1 is the transmission gate. Netlist is shown below. Each transmission gate is composed of a NMOS and a PMOS transistor.

```
subckt Tx VDD VSS Ng Pg in out
parameters size=1
 MN (in Ng out VSS) NMOS size=size
 MP (out Pg in VDD) PMOS size=size
ends Tx
```

Netlist 1 Transmission Gate

16-bit 2:1 Mux and 16-bit 8:1 Mux

In our design review 1, we showed the netlist of our 1-bit 2:1 Mux and 1-bit 8:1 Mux. Netlist 2 and 3 shows 16-bit of those Muxes, as they are needed in the ALU.

```
subckt MUX2_16 VDD VSS\
A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0\
B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0\
s\
O15 O14 O13 O12 O11 O10 O9 O8 O7 O6 O5 O4 O3 O2 O1 O0
parameters size=1
  I15 (VDD VSS A15 B15 s O15) MUX2 size=size
  I14 (VDD VSS A14 B14 s O14) MUX2 size=size
  I13 (VDD VSS A13 B13 s O13) MUX2 size=size
  I12 (VDD VSS A12 B12 s O12) MUX2 size=size
  I11 (VDD VSS A11 B11 s O11) MUX2 size=size
  I10 (VDD VSS A10 B10 s O10) MUX2 size=size
  I9 (VDD VSS A9 B9 s O9) MUX2 size=size
  I8 (VDD VSS A8 B8 s O8) MUX2 size=size
  I7 (VDD VSS A7 B7 s O7) MUX2 size=size
  I6 (VDD VSS A6 B6 s O6) MUX2 size=size
  I5 (VDD VSS A5 B5 s O5) MUX2 size=size
  I4 (VDD VSS A4 B4 s O4) MUX2 size=size
  I3 (VDD VSS A3 B3 s O3) MUX2 size=size
  I2 (VDD VSS A2 B2 s O2) MUX2 size=size
  I1 (VDD VSS A1 B1 s O1) MUX2 size=size
  I0 (VDD VSS A0 B0 s O0) MUX2 size=size
ends MUX2_16
```

Netlist 2 16-bit 2:1 Mux

```
subckt MUX8_16 VDD VSS\
i0n15 i0n14 i0n13 i0n12 i0n11 i0n10 i0n9 i0n8 i0n7 i0n6 i0n5 i0n4 i0n3 i0n2 i0n1 i0n0\
i1n15 i1n14 i1n13 i1n12 i1n11 i1n10 i1n9 i1n8 i1n7 i1n6 i1n5 i1n4 i1n3 i1n2 i1n1 i1n0\
i2n15 i2n14 i2n13 i2n12 i2n11 i2n10 i2n9 i2n8 i2n7 i2n6 i2n5 i2n4 i2n3 i2n2 i2n1 i2n0\
i3n15 i3n14 i3n13 i3n12 i3n11 i3n10 i3n9 i3n8 i3n7 i3n6 i3n5 i3n4 i3n3 i3n2 i3n1 i3n0\
i4n15 i4n14 i4n13 i4n12 i4n11 i4n10 i4n9 i4n8 i4n7 i4n6 i4n5 i4n4 i4n3 i4n2 i4n1 i4n0\
i5n15 i5n14 i5n13 i5n12 i5n11 i5n10 i5n9 i5n8 i5n7 i5n6 i5n5 i5n4 i5n3 i5n2 i5n1 i5n0\
i6n15 i6n14 i6n13 i6n12 i6n11 i6n10 i6n9 i6n8 i6n7 i6n6 i6n5 i6n4 i6n3 i6n2 i6n1 i6n0\
i7n15 i7n14 i7n13 i7n12 i7n11 i7n10 i7n9 i7n8 i7n7 i7n6 i7n5 i7n4 i7n3 i7n2 i7n1 i7n0\
s2 s1 s0\
o15 o14 o13 o12 o11 o10 o9 o8 o7 o6 o5 o4 o3 o2 o1 o0
parameters size=1
 I15 (VDD VSS i7n15 i6n15 i5n15 i4n15 i3n15 i2n15 i1n15 i0n15 s2 s1 s0 o15) MUX8 size=size
 I14 (VDD VSS i7n14 i6n14 i5n14 i4n14 i3n14 i2n14 i1n14 i0n14 s2 s1 s0 o14) MUX8 size=size
 I13 (VDD VSS i7n13 i6n13 i5n13 i4n13 i3n13 i2n13 i1n13 i0n13 s2 s1 s0 o13) MUX8 size=size
 I12 (VDD VSS i7n12 i6n12 i5n12 i4n12 i3n12 i2n12 i1n12 i0n12 s2 s1 s0 o12) MUX8 size=size
 I11 (VDD VSS i7n11 i6n11 i5n11 i4n11 i3n11 i2n11 i1n11 i0n11 s2 s1 s0 o11) MUX8 size=size
 I10 (VDD VSS i7n10 i6n10 i5n10 i4n10 i3n10 i2n10 i1n10 i0n10 s2 s1 s0 o10) MUX8 size=size
 I9 (VDD VSS i7n9 i6n9 i5n9 i4n9 i3n9 i2n9 i1n9 i0n9 s2 s1 s0 o9) MUX8 size=size
```

```
 I8 (VDD VSS i7n8 i6n8 i5n8 i4n8 i3n8 i2n8 i1n8 i0n8 s2 s1 s0 o8) MUX8 size=size
 I7 (VDD VSS i7n7 i6n7 i5n7 i4n7 i3n7 i2n7 i1n7 i0n7 s2 s1 s0 o7) MUX8 size=size
 I6 (VDD VSS i7n6 i6n6 i5n6 i4n6 i3n6 i2n6 i1n6 i0n6 s2 s1 s0 o6) MUX8 size=size
 I5 (VDD VSS i7n5 i6n5 i5n5 i4n5 i3n5 i2n5 i1n5 i0n5 s2 s1 s0 o5) MUX8 size=size
 I4 (VDD VSS i7n4 i6n4 i5n4 i4n4 i3n4 i2n4 i1n4 i0n4 s2 s1 s0 o4) MUX8 size=size
 I3 (VDD VSS i7n3 i6n3 i5n3 i4n3 i3n3 i2n3 i1n3 i0n3 s2 s1 s0 o3) MUX8 size=size
 I2 (VDD VSS i7n2 i6n2 i5n2 i4n2 i3n2 i2n2 i1n2 i0n2 s2 s1 s0 o2) MUX8 size=size
 I1 (VDD VSS i7n1 i6n1 i5n1 i4n1 i3n1 i2n1 i1n1 i0n1 s2 s1 s0 o1) MUX8 size=size
 I0 (VDD VSS i7n0 i6n0 i5n0 i4n0 i3n0 i2n0 i1n0 i0n0 s2 s1 s0 o0) MUX8 size=size
ends MUX8_16
```

Netlist 3 16-bit 8:1 Mux

16-bit Register

Figure 2 shows the topology of the register we built. It is a positive edge-triggered master-slave register.

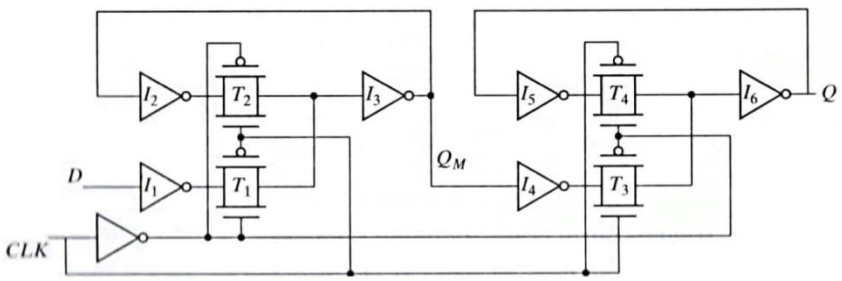

Figure 2 Schematic of Register

Netlist 4 implements Figure 2. Name of the instances are the same as shown in the figure, with I_0 represents the inverter of the clock signal. The 16-bit register instantiated 16 1-bit register (netlist 5).

```
subckt Reg VDD VSS clk D Q
parameters size=1
 I0 (VDD VSS clk clkbar) Inverter size=size
 I1 (VDD VSS D net0) Inverter size=size
 I2 (VDD VSS net3 net1) Inverter size=size
 I3 (VDD VSS net2 net3) Inverter size=size
 I4 (VDD VSS net3 net4) Inverter size=size
 I5 (VDD VSS Q net5) Inverter size=size
 I6 (VDD VSS net6 Q) Inverter size=size

 T1 (VDD VSS clkbar clk net0 net2) Tx size=size
 T2 (VDD VSS clk clkbar net1 net2) Tx size=size
 T3 (VDD VSS clk clkbar net4 net6) Tx size=size
 T4 (VDD VSS clkbar clk net5 net6) Tx size=size
ends Reg
```

Netlist 4 1-bit Register

```
subckt Reg_16 VDD VSS clk\
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0\
Q15 Q14 Q13 Q12 Q11 Q10 Q9 Q8 Q7 Q6 Q5 Q4 Q3 Q2 Q1 Q0
parameters size=1
 I15 (VDD VSS clk D15 Q15) Reg size=size
 I14 (VDD VSS clk D14 Q14) Reg size=size
 I13 (VDD VSS clk D13 Q13) Reg size=size
 I12 (VDD VSS clk D12 Q12) Reg size=size
 I11 (VDD VSS clk D11 Q11) Reg size=size
 I10 (VDD VSS clk D10 Q10) Reg size=size
 I9 (VDD VSS clk D9 Q9) Reg size=size
 I8 (VDD VSS clk D8 Q8) Reg size=size
 I7 (VDD VSS clk D7 Q7) Reg size=size
 I6 (VDD VSS clk D6 Q6) Reg size=size
 I5 (VDD VSS clk D5 Q5) Reg size=size
 I4 (VDD VSS clk D4 Q4) Reg size=size
 I3 (VDD VSS clk D3 Q3) Reg size=size
 I2 (VDD VSS clk D2 Q2) Reg size=size
 I1 (VDD VSS clk D1 Q1) Reg size=size
 I0 (VDD VSS clk D0 Q0) Reg size=size
ends Reg_16
```

Netlist 5 16-bit Register

16-bit Adder-Subtractor

In order to save area, we decided to use only one 16-bit adder to accomplish both adding and subtracting. Since only one type of operation is performed in each clock cycle, this is achievable. The block diagram is shown below in Figure 3. It is a extracted portion of Figure 1.

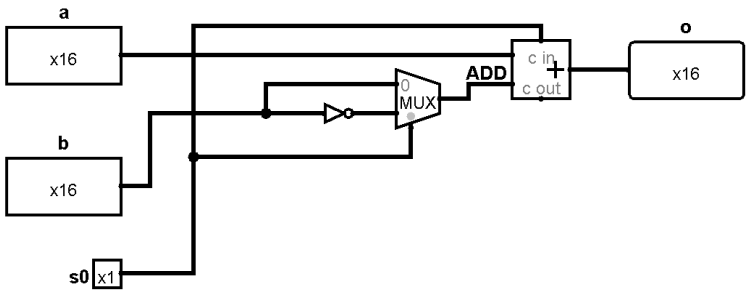

Figure 3 Logic-level schematic of 16-bit Adder-Subtractor

When adding, $s_0 = 0$, the Mux outputs b, and the carry-in to the adder is 0. Therefore,

$$o_+ = a + b$$

When subtracting, $s_0 = 1$, the Mux outputs $\bar{b}$, and the carry-in to the adder is 1. Therefore,

$$o_- = a + \bar{b} + 1$$

By rules of two's complement, $\bar{b} + 1 = -b$, and thus $o\_ = a + (-b) = a - b$

The 16-bit adder is built based on 1-bit mirror adder. The logic-level schematic is shown in Figure 4. Each FA' block is a 1-bit mirror adder that outputs $\overline{C_{out}}$ and $\bar{S}$.

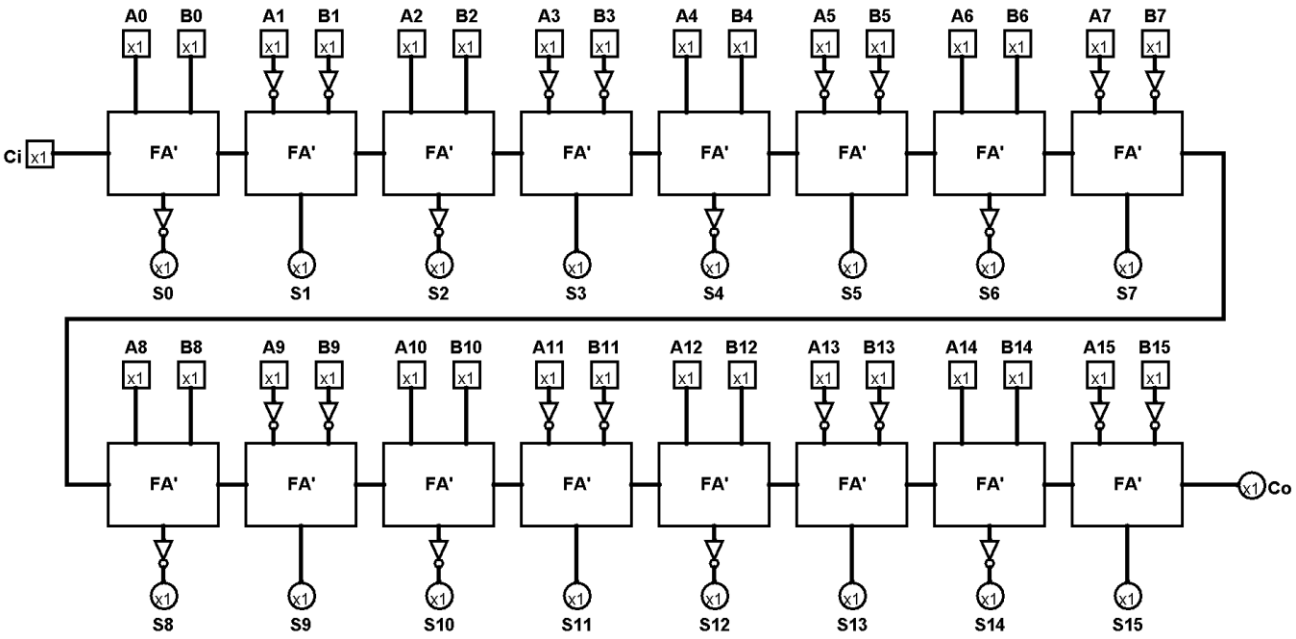

Figure 4 Logic-level schematic of 16-bit Adder

Netlist 6 shows the implementation of each FA' mirror adder block. It was designed and simulated in Problem set 6.

```
subckt FA_inv VDD VSS A B Ci Co_ S_
parameters size=1

 // Carry Stage
 M1 (net0 A VSS VSS) NMOS size=6*size
 M2 (net0 B VSS VSS) NMOS size=6*size
 M3 (Co_ A net1 VSS) NMOS size=2*size
 M4 (net1 B VSS VSS) NMOS size=2*size
 M5 (Co_ Ci net0 VSS) NMOS size=6*size
 M6 (net2 A VDD VDD) PMOS size=12*size
 M7 (net2 B VDD VDD) PMOS size=12*size
 M8 (Co_ A net3 VDD) PMOS size=4*size
 M9 (net3 B VDD VDD) PMOS size=4*size
 M10(Co_ Ci net2 VDD) PMOS size=12*size
```

```
 // Sum Stage
M11 (net4 A VSS VSS) NMOS size=2*size
M12 (net4 B VSS VSS) NMOS size=2*size
M13 (net4 Ci VSS VSS) NMOS size=2*size
M14 (S_ Co_ net4 VSS) NMOS size=2*size
M15 (net5 A VDD VDD) PMOS size=4*size
M16 (net5 B VDD VDD) PMOS size=4*size
M17 (net5 Ci VDD VDD) PMOS size=4*size
M18 (S_ Co_ net5 VDD) PMOS size=4*size

M19 (net6 A VSS VSS) NMOS size=3*size
M20 (net7 B net6 VSS) NMOS size=3*size
M21 (S_ Ci net7 VSS) NMOS size=3*size
M22 (net8 A VDD VDD) PMOS size=6*size
M23 (net9 B net8 VDD) PMOS size=6*size
M24 (S_ Ci net9 VDD) PMOS size=6*size

ends FA_inv
```

Netlist 6 1-bit Mirror Adder

Netlist 7 implements Figure 4.

```
subckt ADD2_16 VDD VSS\
A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0\
B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0\
Ci Co\
S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0
parameters size=1

  A1_inv (VDD VSS A1 A1_) Inverter size=size
  A3_inv (VDD VSS A3 A3_) Inverter size=size
  A5_inv (VDD VSS A5 A5_) Inverter size=size
  A7_inv (VDD VSS A7 A7_) Inverter size=size
  A9_inv (VDD VSS A9 A9_) Inverter size=size
  A11_inv (VDD VSS A11 A11_) Inverter size=size
  A13_inv (VDD VSS A13 A13_) Inverter size=size
  A15_inv (VDD VSS A15 A15_) Inverter size=size

  B1_inv (VDD VSS B1 B1_) Inverter size=size
  B3_inv (VDD VSS B3 B3_) Inverter size=size
  B5_inv (VDD VSS B5 B5_) Inverter size=size
  B7_inv (VDD VSS B7 B7_) Inverter size=size
  B9_inv (VDD VSS B9 B9_) Inverter size=size
```

```
  B11_inv (VDD VSS B11 B11_) Inverter size=size
  B13_inv (VDD VSS B13 B13_) Inverter size=size
  B15_inv (VDD VSS B15 B15_) Inverter size=size

  S0_inv (VDD VSS S0_ S0) Inverter size=size
  S2_inv (VDD VSS S2_ S2) Inverter size=size
  S4_inv (VDD VSS S4_ S4) Inverter size=size
  S6_inv (VDD VSS S6_ S6) Inverter size=size
  S8_inv (VDD VSS S8_ S8) Inverter size=size
  S10_inv (VDD VSS S10_ S10) Inverter size=size
  S12_inv (VDD VSS S12_ S12) Inverter size=size
  S14_inv (VDD VSS S14_ S14) Inverter size=size

  I0 (VDD VSS A0 B0 Ci C0 S0_) FA_inv size=1
  I1 (VDD VSS A1_ B1_ C0 C1 S1) FA_inv size=1
  I2 (VDD VSS A2 B2 C1 C2 S2_) FA_inv size=1
  I3 (VDD VSS A3_ B3_ C2 C3 S3) FA_inv size=1
  I4 (VDD VSS A4 B4 C3 C4 S4_) FA_inv size=1
  I5 (VDD VSS A5_ B5_ C4 C5 S5) FA_inv size=1
  I6 (VDD VSS A6 B6 C5 C6 S6_) FA_inv size=1
  I7 (VDD VSS A7_ B7_ C6 C7 S7) FA_inv size=1
  I8 (VDD VSS A8 B8 C7 C8 S8_) FA_inv size=1
  I9 (VDD VSS A9_ B9_ C8 C9 S9) FA_inv size=1
  I10 (VDD VSS A10 B10 C9 C10 S10_) FA_inv size=1
  I11 (VDD VSS A11_ B11_ C10 C11 S11) FA_inv size=1
  I12 (VDD VSS A12 B12 C11 C12 S12_) FA_inv size=1
  I13 (VDD VSS A13_ B13_ C12 C13 S13) FA_inv size=1
  I14 (VDD VSS A14 B14 C13 C14 S14_) FA_inv size=1
  I15 (VDD VSS A15_ B15_ C14 Co S15) FA_inv size=1
ends ADD2_16
```

Netlist 7 16-bit Ripple Carry Adder based on 1-bit Mirror Adder

Netlist 8 implements Figure 3. When $control = 0,\ S = A + B$. Otherwise when $control = 1,\ S = A - B$.

```
subckt ADDSUB_16 VDD VSS\
A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0\
B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0\
control Co\
S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0
parameters size=1

  adder (VDD VSS\
    A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0\
```

```
    b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0\
    control Co\
    S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0) ADD2_16 size=size

  mux (VDD VSS\
    B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0\
    B15_ B14_ B13_ B12_ B11_ B10_ B9_ B8_ B7_ B6_ B5_ B4_ B3_ B2_ B1_ B0_\
    control\
    b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0) MUX2_16 size=size

  Inverter (VDD VSS\
    B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0\
    B15_ B14_ B13_ B12_ B11_ B10_ B9_ B8_ B7_ B6_ B5_ B4_ B3_ B2_ B1_ B0_) Inverter_16 size=size

ends ADDSUB_16
```

Netlist 8 16-bit Adder-Subtractor

16-bit Shifter

The 16-bit shifter is a barrel shifter, as shown in netlist 9.

In the first stage, input a is either shifted 2 bits or not shifted if b_1 is 1 or 0, respectively. The output of the first stage is 16-bit x. In the second stage, x is shifted either 2 bits or 1 bit if b_1 is 1 or 0, respectively. The output of the second stage is 16-bit y.

We used pass gates to implement the shifter so buffers are needed to recover the signal swing.

```
subckt Shifter VDD VSS \
a15 a14 a13 a12 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0 \
b1 b0 \
o15 o14 o13 o12 o11 o10 o9 o8 o7 o6 o5 o4 o3 o2 o1 o0
parameters size=1

  // Signal b0_ and b1_
  Inv_b1 (VDD VSS b1 b1_) Inverter size=size
  Inv_b0 (VDD VSS b0 b0_) Inverter size=size

  // When b1 = 1, shift left by 2.
  I15 (a13 b1 x15 VSS) NMOS size=size
  I14 (a12 b1 x14 VSS) NMOS size=size
  I13 (a11 b1 x13 VSS) NMOS size=size
  I12 (a10 b1 x12 VSS) NMOS size=size
```

```
I11 (a9 b1 x11 VSS) NMOS size=size
I10 (a8 b1 x10 VSS) NMOS size=size
I9 (a7 b1 x9 VSS) NMOS size=size
I8 (a6 b1 x8 VSS) NMOS size=size
I7 (a5 b1 x7 VSS) NMOS size=size
I6 (a4 b1 x6 VSS) NMOS size=size
I5 (a3 b1 x5 VSS) NMOS size=size
I4 (a2 b1 x4 VSS) NMOS size=size
I3 (a1 b1 x3 VSS) NMOS size=size
I2 (a0 b1 x2 VSS) NMOS size=size
I1 (VSS b1 x1 VSS) NMOS size=size
I0 (VSS b1 x0 VSS) NMOS size=size

// When b1 = 0, no shift.
I15_ (a15 b1_ x15 VSS) NMOS size=size
I14_ (a14 b1_ x14 VSS) NMOS size=size
I13_ (a13 b1_ x13 VSS) NMOS size=size
I12_ (a12 b1_ x12 VSS) NMOS size=size
I11_ (a11 b1_ x11 VSS) NMOS size=size
I10_ (a10 b1_ x10 VSS) NMOS size=size
I9_ (a9 b1_ x9 VSS) NMOS size=size
I8_ (a8 b1_ x8 VSS) NMOS size=size
I7_ (a7 b1_ x7 VSS) NMOS size=size
I6_ (a6 b1_ x6 VSS) NMOS size=size
I5_ (a5 b1_ x5 VSS) NMOS size=size
I4_ (a4 b1_ x4 VSS) NMOS size=size
I3_ (a3 b1_ x3 VSS) NMOS size=size
I2_ (a2 b1_ x2 VSS) NMOS size=size
I1_ (a1 b1_ x1 VSS) NMOS size=size
I0_ (a0 b1_ x0 VSS) NMOS size=size

// When b0 = 1, shift left by 2.
II15 (x13 b0 y15 VSS) NMOS size=size
II14 (x12 b0 y14 VSS) NMOS size=size
II13 (x11 b0 y13 VSS) NMOS size=size
II12 (x10 b0 y12 VSS) NMOS size=size
II11 (x9 b0 y11 VSS) NMOS size=size
II10 (x8 b0 y10 VSS) NMOS size=size
II9 (x7 b0 y9 VSS) NMOS size=size
II8 (x6 b0 y8 VSS) NMOS size=size
II7 (x5 b0 y7 VSS) NMOS size=size
II6 (x4 b0 y6 VSS) NMOS size=size
II5 (x3 b0 y5 VSS) NMOS size=size
II4 (x2 b0 y4 VSS) NMOS size=size
```

```
  II3 (x1 b0 y3 VSS) NMOS size=size
  II2 (x0 b0 y2 VSS) NMOS size=size
  II1 (VSS b0 y1 VSS) NMOS size=size
  II0 (VSS b0 y0 VSS) NMOS size=size

  // When b0 = 0, shift left by 1.
  II15_ (x14 b0_ y15 VSS) NMOS size=size
  II14_ (x13 b0_ y14 VSS) NMOS size=size
  II13_ (x12 b0_ y13 VSS) NMOS size=size
  II12_ (x11 b0_ y12 VSS) NMOS size=size
  II11_ (x10 b0_ y11 VSS) NMOS size=size
  II10_ (x9 b0_ y10 VSS) NMOS size=size
  II9_ (x8 b0_ y9 VSS) NMOS size=size
  II8_ (x7 b0_ y8 VSS) NMOS size=size
  II7_ (x6 b0_ y7 VSS) NMOS size=size
  II6_ (x5 b0_ y6 VSS) NMOS size=size
  II5_ (x4 b0_ y5 VSS) NMOS size=size
  II4_ (x3 b0_ y4 VSS) NMOS size=size
  II3_ (x2 b0_ y3 VSS) NMOS size=size
  II2_ (x1 b0_ y2 VSS) NMOS size=size
  II1_ (x0 b0_ y1 VSS) NMOS size=size
  II0_ (VSS b0_ y0 VSS) NMOS size=size

  // Buffer the outputs to restore the voltage swing
  Buf15 (VDD VSS y15 o15) Buffer size=size
  Buf14 (VDD VSS y14 o14) Buffer size=size
  Buf13 (VDD VSS y13 o13) Buffer size=size
  Buf12 (VDD VSS y12 o12) Buffer size=size
  Buf11 (VDD VSS y11 o11) Buffer size=size
  Buf10 (VDD VSS y10 o10) Buffer size=size
  Buf9 (VDD VSS y9 o9) Buffer size=size
  Buf8 (VDD VSS y8 o8) Buffer size=size
  Buf7 (VDD VSS y7 o7) Buffer size=size
  Buf6 (VDD VSS y6 o6) Buffer size=size
  Buf5 (VDD VSS y5 o5) Buffer size=size
  Buf4 (VDD VSS y4 o4) Buffer size=size
  Buf3 (VDD VSS y3 o3) Buffer size=size
  Buf2 (VDD VSS y2 o2) Buffer size=size
  Buf1 (VDD VSS y1 o1) Buffer size=size
  Buf0 (VDD VSS y0 o0) Buffer size=size
ends Shifter
```

Netlist 9 16-bit Shifter

Top-level Design

Netlist 10 implements top-level design, as shown in Figure 1.

For the adder-subtractor, s_0 is wired to the control input of the adder-subtractor. This is made possible by our arrangement of op-code; ADD is 100 and SUB is 101. The least significant control bit is sufficient to select determine the behavior of the add-subtractor.

```
regA (vdd! 0 clk\
  A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0\
  a15 a14 a13 a12 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0) Reg_16 size=2

regB (vdd! 0 clk\
  B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5 B4 B3 B2 B1 B0\
  b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0) Reg_16 size=2

regO (vdd! 0 clk\
  o15 o14 o13 o12 o11 o10 o9 o8 o7 o6 o5 o4 o3 o2 o1 o0\
  O15 O14 O13 O12 O11 O10 O9 O8 O7 O6 O5 O4 O3 O2 O1 O0) Reg_16 size=2

regS2 (vdd! 0 clk S2 s2) Reg size=8
regS1 (vdd! 0 clk S1 s1) Reg size=8
regS0 (vdd! 0 clk S0 s0) Reg size=8

and (vdd! 0\
  a15 a14 a13 a12 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0\
  b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0\
  i1n15 i1n14 i1n13 i1n12 i1n11 i1n10 i1n9 i1n8 i1n7 i1n6 i1n5 i1n4 i1n3 i1n2 i1n1 i1n0) AND2_16

or (vdd! 0\
  a15 a14 a13 a12 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0\
  b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0\
  i2n15 i2n14 i2n13 i2n12 i2n11 i2n10 i2n9 i2n8 i2n7 i2n6 i2n5 i2n4 i2n3 i2n2 i2n1 i2n0) OR2_16

pass (vdd! 0\
  a15 a14 a13 a12 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0\
  i3n15 i3n14 i3n13 i3n12 i3n11 i3n10 i3n9 i3n8 i3n7 i3n6 i3n5 i3n4 i3n3 i3n2 i3n1 i3n0) Pass

addsub (vdd! 0\
  a15 a14 a13 a12 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0\
  b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0\
  s0 Co\
  i45n15 i45n14 i45n13 i45n12 i45n11 i45n10 i45n9 i45n8 i45n7 i45n6 i45n5 i45n4 i45n3 i45n2 i45n1 i45n0)
```

```
ADDSUB_16

shift (vdd! 0\
  a15 a14 a13 a12 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0 \
  b1 b0 \
  i6n15 i6n14 i6n13 i6n12 i6n11 i6n10 i6n9 i6n8 i6n7 i6n6 i6n5 i6n4 i6n3 i6n2 i6n1 i6n0) Shifter

////////////////////////////// ARBITARY HERE ////////////////////////////////
passb7 (vdd! 0\
  a15 a14 a13 a12 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0\
  i7n15 i7n14 i7n13 i7n12 i7n11 i7n10 i7n9 i7n8 i7n7 i7n6 i7n5 i7n4 i7n3 i7n2 i7n1 i7n0) Pass
/////////////////////////////////////////////////////////////////////////////

alumux (vdd! 0\
  O15 O14 O13 O12 O11 O10 O9 O8 O7 O6 O5 O4 O3 O2 O1 O0\
  i1n15 i1n14 i1n13 i1n12 i1n11 i1n10 i1n9 i1n8 i1n7 i1n6 i1n5 i1n4 i1n3 i1n2 i1n1 i1n0\
  i2n15 i2n14 i2n13 i2n12 i2n11 i2n10 i2n9 i2n8 i2n7 i2n6 i2n5 i2n4 i2n3 i2n2 i2n1 i2n0\
  i3n15 i3n14 i3n13 i3n12 i3n11 i3n10 i3n9 i3n8 i3n7 i3n6 i3n5 i3n4 i3n3 i3n2 i3n1 i3n0\
  i45n15 i45n14 i45n13 i45n12 i45n11 i45n10 i45n9 i45n8 i45n7 i45n6 i45n5 i45n4 i45n3 i45n2 i45n1 i45n0\
  i45n15 i45n14 i45n13 i45n12 i45n11 i45n10 i45n9 i45n8 i45n7 i45n6 i45n5 i45n4 i45n3 i45n2 i45n1 i45n0\
  i6n15 i6n14 i6n13 i6n12 i6n11 i6n10 i6n9 i6n8 i6n7 i6n6 i6n5 i6n4 i6n3 i6n2 i6n1 i6n0\
  i7n15 i7n14 i7n13 i7n12 i7n11 i7n10 i7n9 i7n8 i7n7 i7n6 i7n5 i7n4 i7n3 i7n2 i7n1 i7n0\
  s2 s1 s0\
  o15 o14 o13 o12 o11 o10 o9 o8 o7 o6 o5 o4 o3 o2 o1 o0) MUX8_16
```

Netlist 10 Top-Level Design

Design Verification

16-bit Register

The 16-bit register is simulated for 3.5 clock cycles. Clock period is 500 ps.

The value of the input signal D at the three rising edges are 0, 1 and 0, respectively, and therefore the output signal Q should be high during the second clock cycle, and low during the first and the third clock cycle.

Figure 5 shows the simulation results of Bit 7, and the plot of Q matches the expectation and thus verifies the design. We made sure that all the other bits also functions correctly.

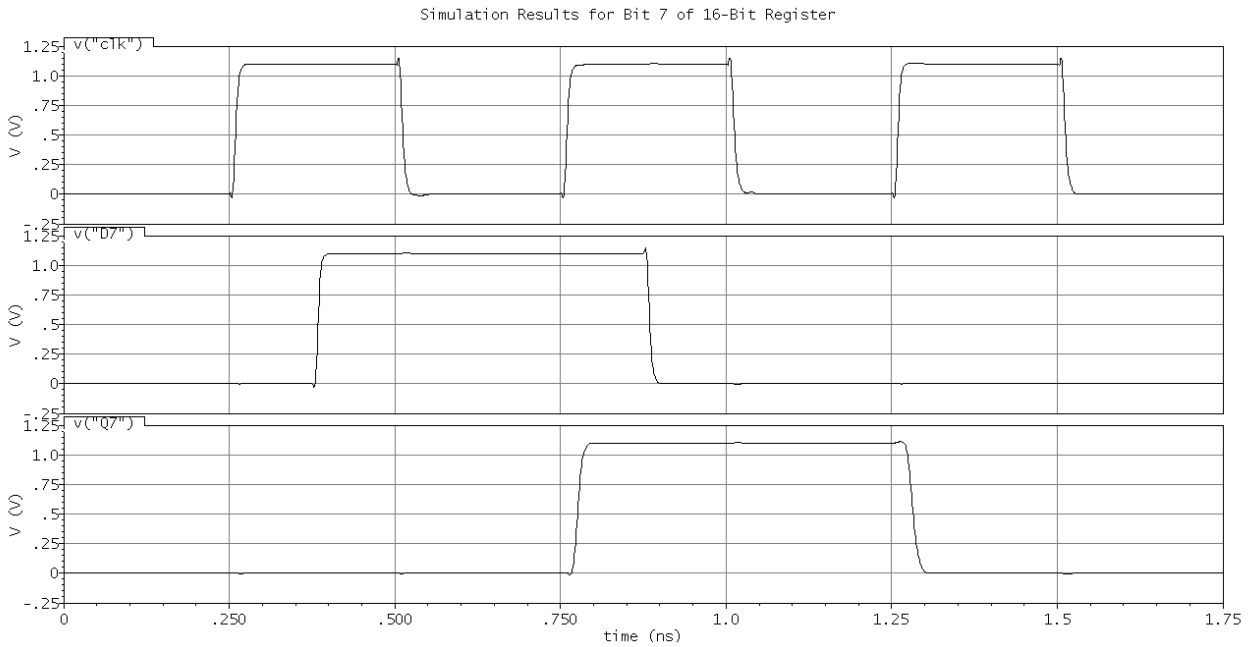

Figure 5 Simulation results of the 16-bit Register @ Bit 7- 1.75ns (500ps/cycle)

16-bit Adder-Subtractor

The 16-bit adder-subtractor is simulated for 2 periods. Data inputs A and B are assigned fixed values.

$$A = \text{0b } 1011\ 0000\ 0001\ 1111$$

$$B = \text{0b } 1111\ 0011\ 0100\ 1010$$

The control bits changes from 1 to 0 (Figure 6). Therefore, the unit perform subtraction in the first period and addition in the second period.

The expected outputs in the first period is:

$$S = A - B = A + \bar{B} + 1 = S = \text{0b } (0)\ 1011\ 1100\ 1101\ 0101$$

The "0" in the parenthesis is the carry-out bit.

The expected outputs in the second period is:

$$S = A + B = \text{0b } (1)\ 1010\ 0011\ 0110\ 1001$$

The simulated output signal shown in Figure 7 and 8 matches the expected values.

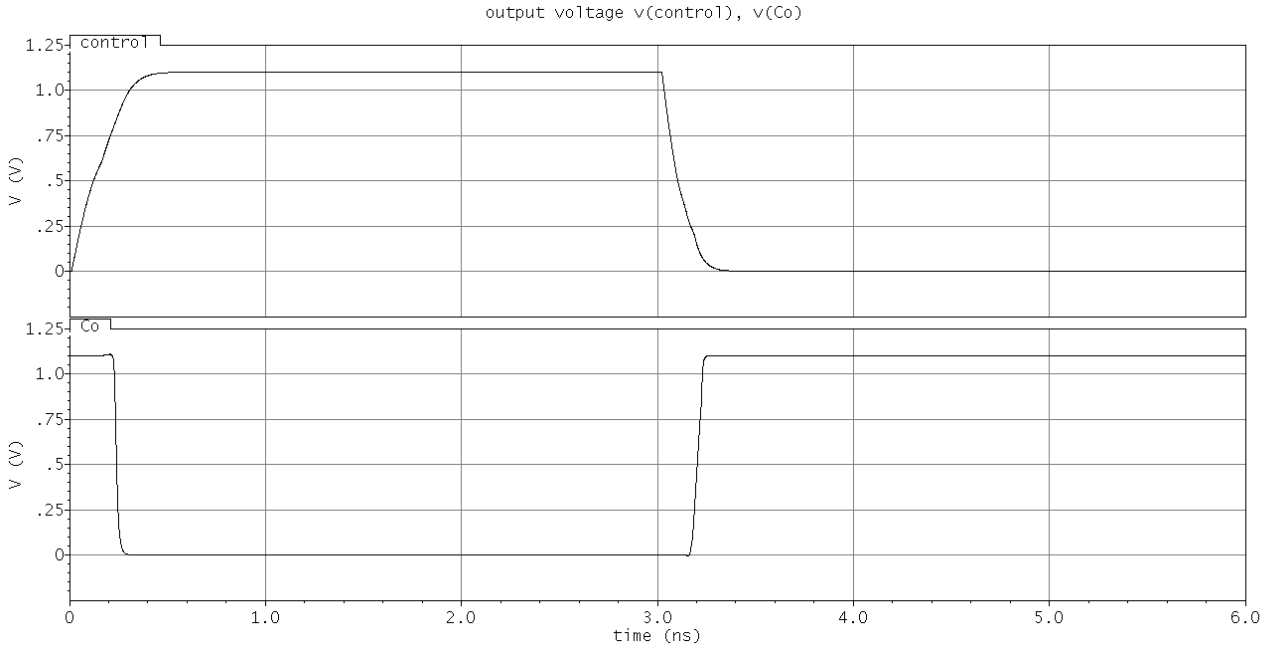

Figure 6 Simulation results of 16-bit Adder-Subtractor – 6ns: $control,\ C_o$

16-bit Shifter

The 16-bit shifter is simulated for 4 periods. The value of the data input A is kept constant for all the 4 periods. Shift control bits (B<1:0>) is assigned one value (among 00, 01, 10, and 11) for each cycle.

The value assigned to A is

$$A = \text{0b } 1110\ 0110\ 1010\ 0111$$

So the outputs of the 4 cycles are expected to be:

$$O_1 = \text{0b } 1110\ 0110\ 1010\ 0111$$
$$O_2 = \text{0b } 1100\ 1101\ 0100\ 1110$$
$$O_3 = \text{0b } 1001\ 1010\ 1001\ 1100$$
$$O_4 = \text{0b } 0011\ 0101\ 0011\ 1000$$

Figure 9 shows the control signals and Figure 10 and 11 shows the data output.

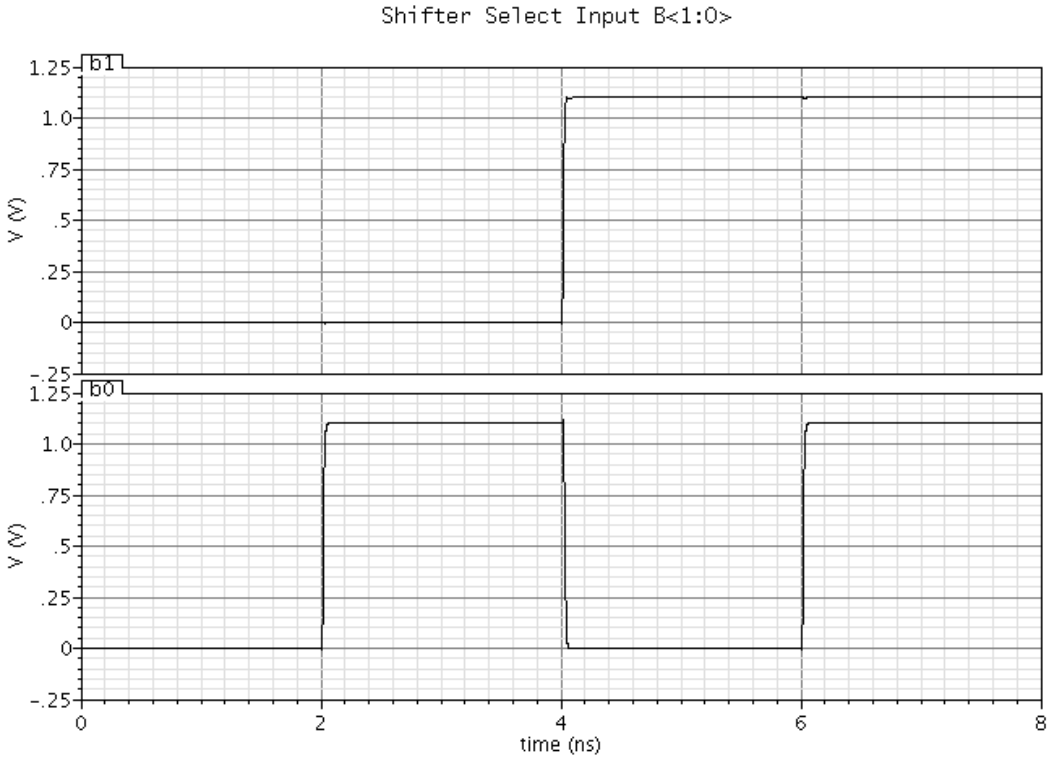

Figure 9 Simulation results of 16-Shifter – 8ns: B<1:0>

Top-Level Design

The top-level circuit is simulated for 20 ns. Each clock cycle is 2 ns. Data input A and B are set to fixed values, while the control signals sweep from S<2:0>= 111 → 000.

Figure 10 and 11 shows that the control signal are corrected latched by the registers. Figure 10 shows signal S<2:0>, which is the input signal to the DSP system. Figure 11 shows signal s<2:0>, which is the output signal (latches signal) of the positive edge-triggered registers.

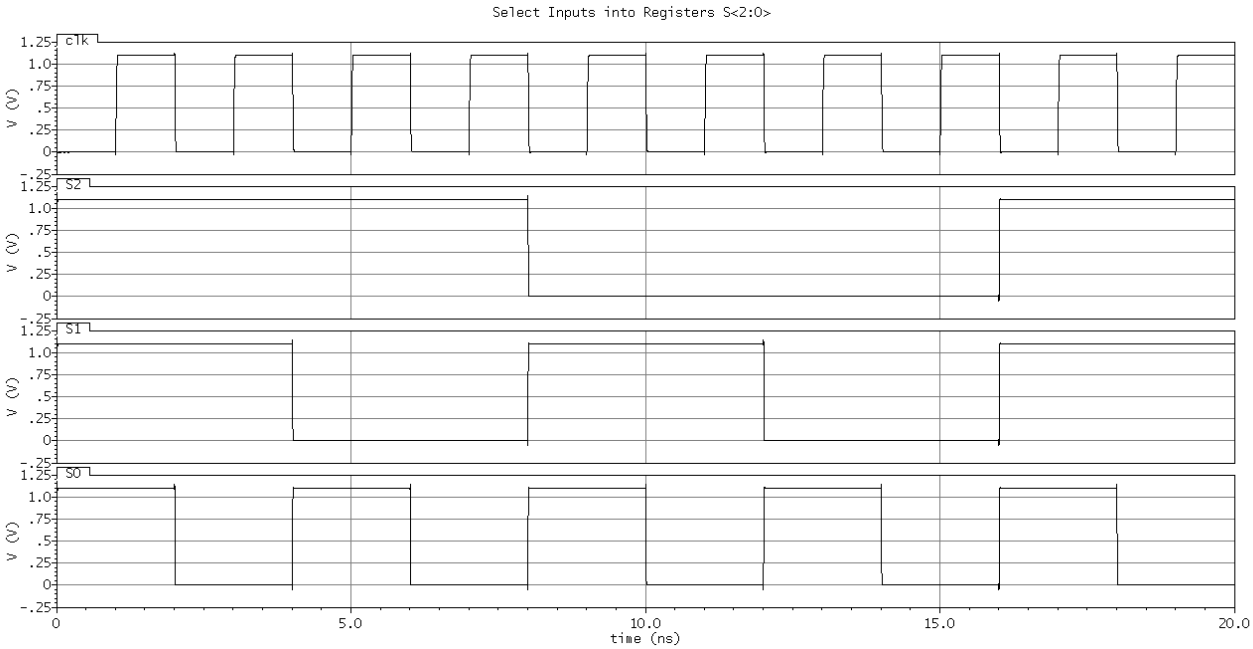

Figure 10 Simulation results of the ALU – 20ns: S<2:0>

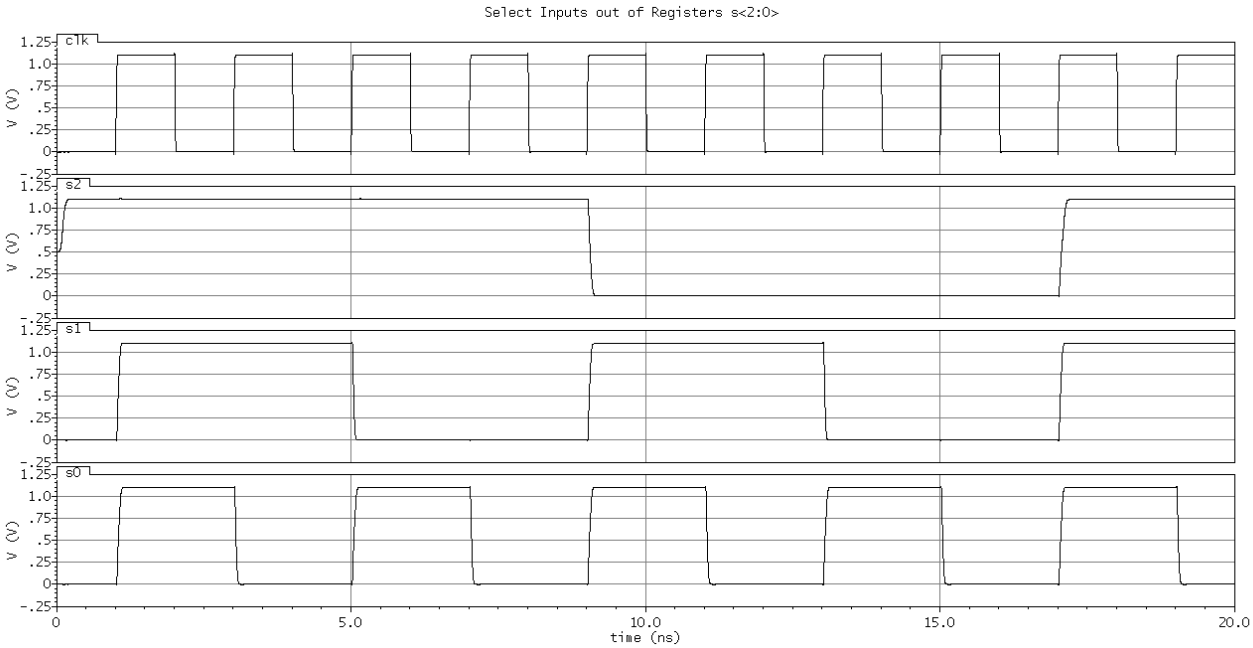

Figure 11 Simulation results of the ALU – 20ns: s<2:0>

The assigned data input values are:

$$A = 0x08A7 = 2215$$

$$B = 0xFEA7 = -345$$

We predict and observe signal O<15:0>, which is the output of the register. Since both inputs and outputs are connected to registers, the value of O after i^{th} rising edge will be the result of the $(i-1)^{th}$ operation.

In the 2^{nd} cycle, O shows the result of $S = 0\text{b } 111$ (PASS A)

$$O = A = 0x08A7$$

In the 3^{rd} cycle, O shows the result of $S = 0\text{b } 110$ (SHIFT)

$$O = A \ll 4 = 0x8A70$$

In the 4^{th} cycle, O shows the result of $S = 0\text{b } 101$ (SUB)

$$O = A - B = 2215 - (-345) = 0x0A00$$

In the 5^{th} cycle, O shows the result of $S = 0\text{b } 100$ (ADD)

$$O = A + B = 2215 + (-345) = 0x074E$$

In the 6^{th} cycle, O shows the result of $S = 0\text{b } 011$ (PASS A)

$$O = A = 0x08A7$$

In the 7^{th} cycle, O shows the result of $S = 0\text{b } 010$ (OR)

$$O = A|B = 0xFFA7$$

In the 8^{th} cycle, O shows the result of $S = 0\text{b } 001$ (AND)

$$O = A\&B = 0x08A7$$

In the 9^{th} cycle, O shows the result of $S = 0\text{b } 000$ (NOP)

$$O = 0X08A7$$

Note that the i^{th} cycles stands for the time period between the i^{th} rising edge and the $(i+1)^{th}$ rising edge.

Figure 12-15 verifies the design.

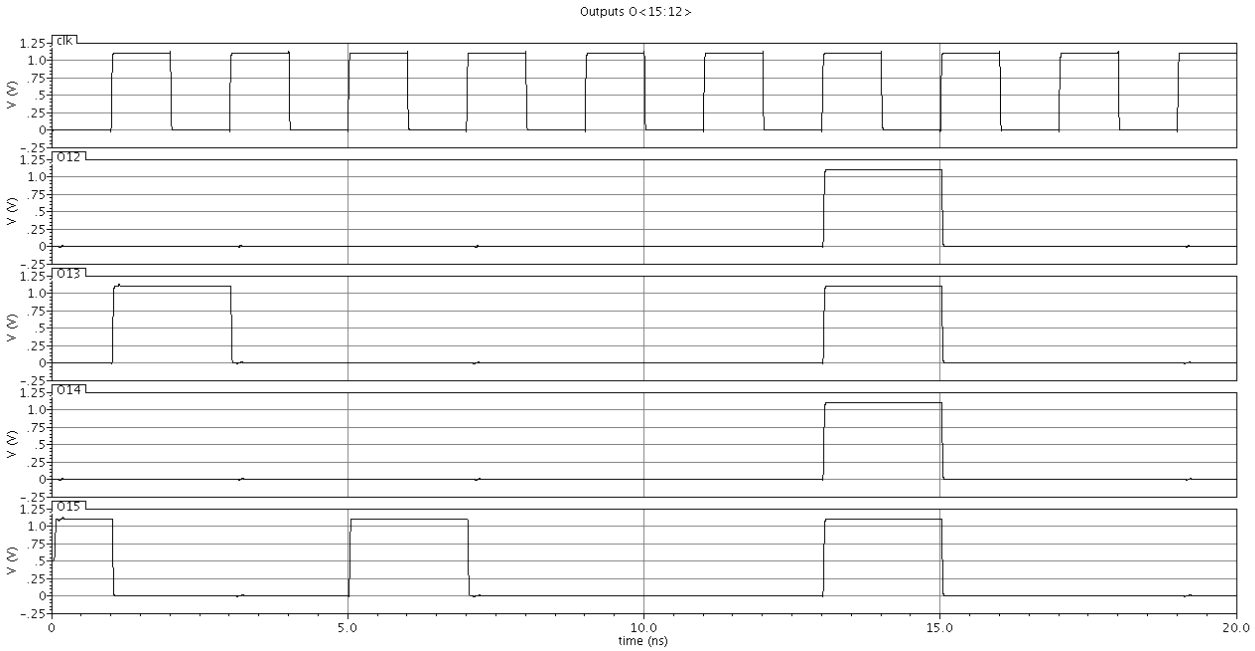

Figure 12 Simulation results of the ALU – 20ns: *O*<15:12>

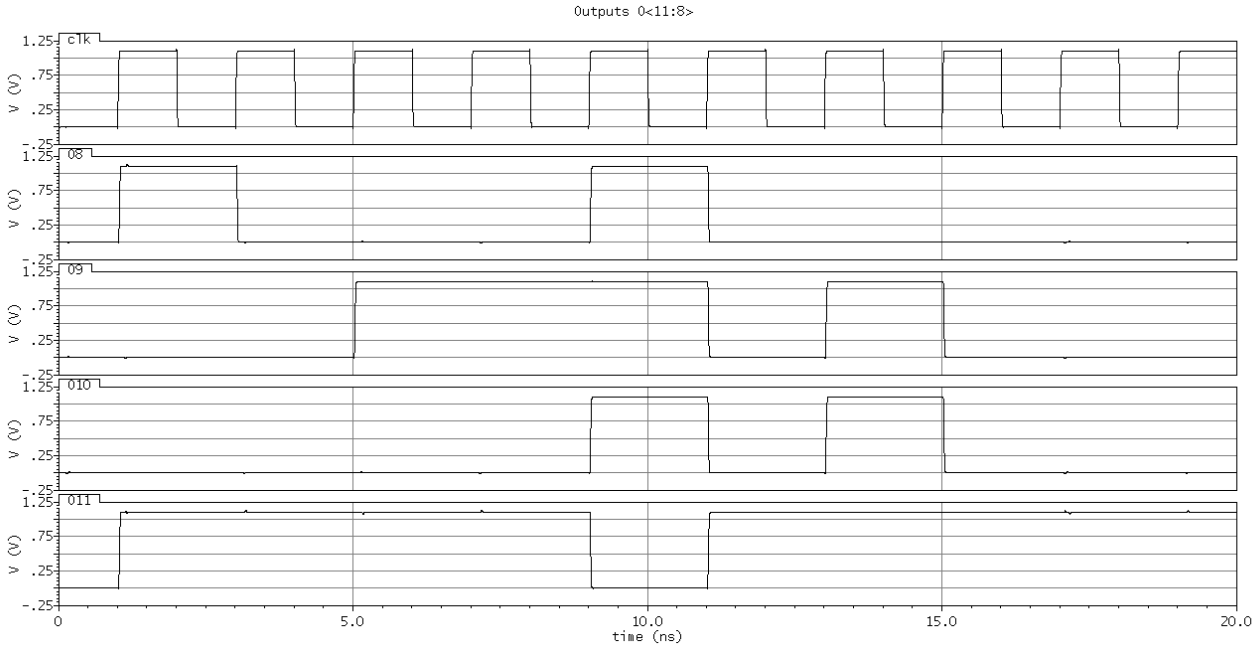

Figure 13 Simulation results of the ALU – 20ns: *O*<11:8>

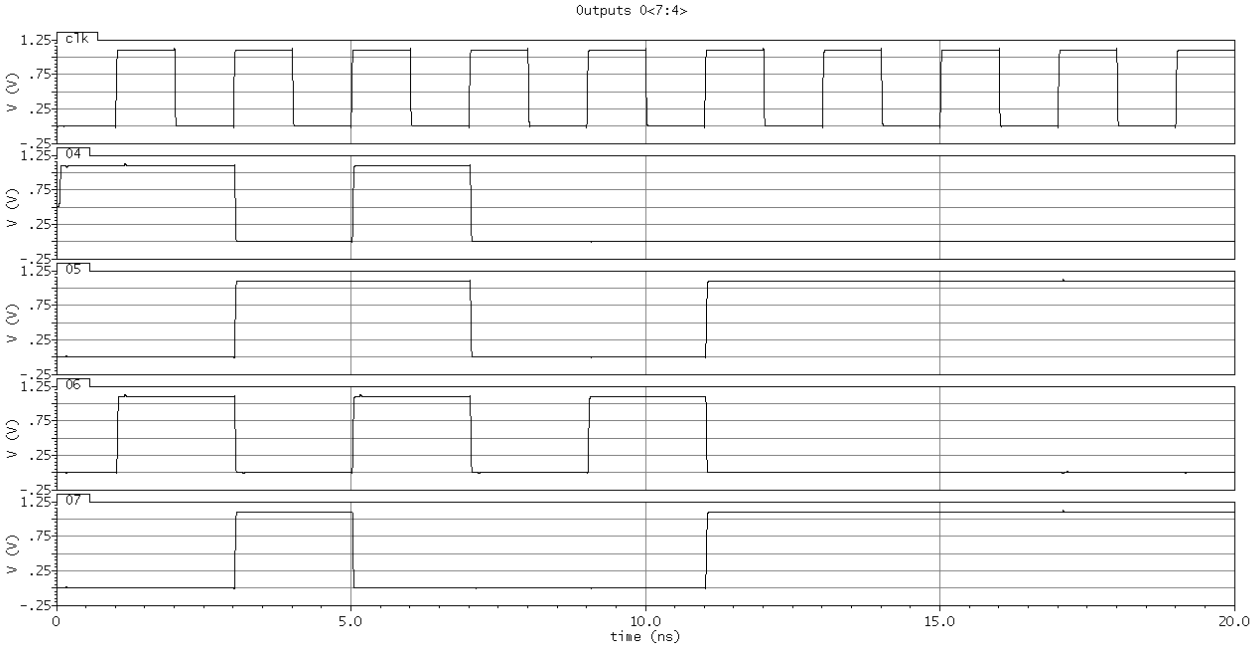

Figure 14 Simulation results of the ALU – 20ns: *O*<7:4>

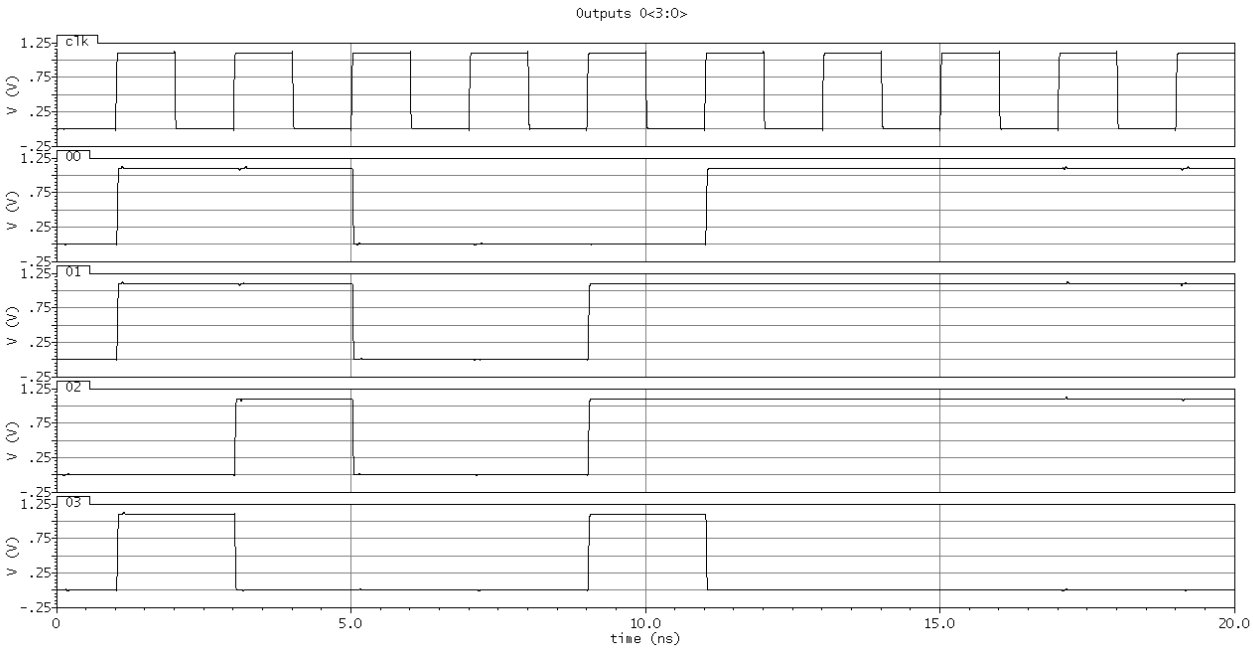

Figure 15 Simulation results of the ALU – 20ns: *O*<7:4>

Design Improvement

PASS A

In our design review 1, we used one buffer for each bit to accomplish PASS A. We now switched to use one transmission gate per bit to accomplish the same task. This saves 2 transistors per bit without affecting overall performance of the circuit.

Delay

ADD

Worst case delay is achieved when

$$A = 0x7\text{FFF}$$

$$B = 0x0000 \rightarrow 0x0001$$

When $B = 0x0000$, $A + B = 0x7\text{FFF}$, each bit produces a 0 carry-out.

When transition happens, the least significant adds up to 0, and produces a carry-out. Thereafter, each bit carry-out a "1" to the bit on its left, and thus the carry ripples towards the MSB, where the sum (S_{15}) changes. This chain takes 1 t_{mux}, 15 t_{cout} and 1 t_{sum}. The final output after the transition should be

$$S = 0x8000$$

Simulation shows that

$$t_{p,ADD} = 298.86 \text{ ps}$$

SUB

Similar to ADD,

$$A = 0x0000$$

$$B = 0x8001 \rightarrow 0x8000$$

The transition is equivalent to

$$\bar{B} = 0x7\text{FFE} \rightarrow 0x7\text{FFF}$$

Before the transition $S = A + \bar{B} + 1 = 0x7\text{FFFF}$. No carry-out is generated for any bit.

After the transition, carry-out is rippled from the LSB to the MSB, incurring a worst-case delay.

$$S = 0x8000$$

$$t_{p,SUB} = 274.62 \text{ ps}$$

SHIFT

Except for that logic 0 might be shifted into the lowest several bits, the path for the input to output consists of 2 pass gates (NMOS transistors) and a buffer, and this is the critical path. For the worst case scenario, the value passed by or the state of both pass gate changes.

We assign $A = 0x0100$, namely **Bit 8** is a **1** and the rest are **0**s. We make B<1:0> transit from 00 to 11. This is the worst case scenario since both pass gates were off and are turned on at the transition. Also, NMOS passes a weak 1, so slower passing is expected. We measured the delay from when the transition occurs to when **Bit 12** changes from 0 to 1.

$$t_p = 68.90 \text{ ps}$$

AND/OR/PASS A

Each bit of the AND, OR and PASS A functions the same, so it is suffice to only measure the delay of one bit. Therefore, we measured worst case t_{pLH} and worst case t_{pHL}, and choose the longer one as the worst case propagation delay.

For **AND** gate ($Out = A \,\&\, B$):

- Worst case high-to-low transition is $A_iB_i = 11 \rightarrow 10,\ t_{pHL} = 20.03$ ps
- Worst case low-to-high transition is $A_iB_i = 10 \rightarrow 11,\ t_{pLH} = 22.74$ ps
- Therefore, the longest-delay scenario is $A_iB_i = 10 \rightarrow 11$.

For **OR** gate ($Out = A \mid B$):

- Worst case high-to-low transition is $A_iB_i = 10 \rightarrow 00,\ t_{pHL} = 23.16$ ps
- Worst case low-to-high transition is $A_iB_i = 00 \rightarrow 10,\ t_{pLH} = 18.77$ ps
- Therefore, the longest-delay scenario is $A_iB_i = 10 \rightarrow 00$

For **PASS A** ($Out = A$):

- Worst case high-to-low transition is $A_i = 1 \rightarrow 0,\ t_{pHL} = 11.10$ ps
- Worst case low-to-high transition is $A_i = 0 \rightarrow 1,\ t_{pLH} = 10.30$ ps
- Therefore, the longest-delay scenario is $A_i = 1 \rightarrow 0$

Summary

Table 2 Summary of worst-case propagation delay of combinational components

Operation	Worst-Case Delay	Scenario
ADD	298.86 ps	$A = 0x7FFF,\ B = 0x0000 \rightarrow 0x0001$
SUB	274.62ps	$A = 0x0000,\ B = 0x8001 \rightarrow 0x8000$
SHIFT	68.90 ps	$A = 0x0100,\ B$<1:0>=00 $\rightarrow 11$
AND	22.74 ps	$A_iB_i = 10 \rightarrow 11$
OR	23.16 ps	$A_iB_i = 10 \rightarrow 00$
PASS A	11.10 ps	$A_i = 1 \rightarrow 0$

Progress and Task Remained

Up till design review 2, we have completed wiring up the top-level circuit for the ALU and verified the correctness of the design. Functions fulfilled are including NOP, AND, OR, PASS A, ADD, SUB, and SHIFT.

For the ALU design, we decided to feed the select bits S into registers, because by doing so, the entire structure can be treated as one stage in a pipeline. This could be beneficial if the circuit we built is a portion of a larger digital system.

The two remaining tasks are the arbitrary function and optimization. Prof. Calhoun agreed that our arbitrary function can be a Multiplier, which will multiply two 8-bit numbers (A<7:0> and B<7:0>) and have a 16-bit output. We believe this is more beneficial than having a 16-bit multiplier, because the delay of that component could really skew the average delay of all components.

The metric that we must minimize as much as possible is $Metric = (Active\ Power) * Delay^2 * Area$. For our circuit performance, there are many things we can optimize.

In terms of delay, we could use logical efforts to calculate the optimum sizing of each component. In terms of area, we could implement pass gate transistor logic to replace static CMOS logic. This could greatly reduce the number of transistors needed. The Adder could be turned into a Manchester Carry Chain, and because early on we made the decision to use the Adder with for both the Add and Subtract functions, we only need to implement one adder instead of keeping them separate and dealing with two. The Multiplier could be built as a Wallace Tree Multiplier which would distribute the weight of anding and adding bits across the component, reducing delay and area. The registers could also be implemented with two half Muxes, but that may be tricky since we'll have to be careful not to fall into the metastable region. In order to minimize active power, we could reduce the amount of voltage our components need to function properly, because voltage is quadratically proportional to dynamic power.